**Problem Statement:**

Submitted by: Bhuwanesh Nainwal

Diwali is the festival of lights. It is one of the biggest and grandest festivals celebrated mainly in India. During night some people also light up rockets (a kind of firecracker) in the sky. Each of the rockets remains in the sky for a specific duration and not necessarily the same. On the day of Diwali, total n rockets were lit up by people where n is a positive integer. You are given the instant (an integer) at which rockets will leave the ground and the instant (an integer) at which they will come back to the ground again. You have to find the maximum number of rockets that would be in the sky at the same time.

Note that at the instant of start-time (**Timestart**), rockets will be considered above the ground and at the instant of end-time (**Timeend**) also the rocket would considered be above the ground.

**Problem based on greedy algorithm:**

**Input format:**

The first line of the input consists of a positive integer n.

Next n lines consist two non-negative integers in each of the lines.

**Output format:**

Print a positive integer: maximum number of rockets that would be in the sky at the same time

**Constraints:**

**Constraints 1 (25 points):**

**1 <= N <= 103, 0 <= Timestart <= Timeend <= 104**

**Constraints 2 (100 Points):**

**1 <= N <= 107, 0 <= Timestart < = Timeend <= 1018**

**Sample test cases:**

**Sample Input 1:**

[ {2, 4}, {3, 5}, {10, 18}, {17, 28}, {15, 19} ]

**Sample Output 1:**

3

**Explanation:** Rockets at positions 3rd, 4th and 5th from the beginning would be in the sky at the same time for the instant 17th.

**Sample Input 2:**

[ {7, 14}, {19, 26}, {28, 32}, {25, 27} ]

**Sample Output 2:**

2

**Explanation:** Rockets at positions 2nd and 4th from the beginning would be in the sky at the same time for the instant 25th.

**Editorial:**

**1.** **Naive approach:**

**Hint**  : Brute force and check for each instant of time one by one.

**Intuition** 💡 : There exists at least one instant of time at which maximum number of rockets would be in the sky.

**Explanation**  : Let the maximum end-time out of all the rockets is denoted by **maxEndTime**. We can initialize a global variable with the minimum value like 0 that store our final answer .We will iterate from 1 to **maxEndTime** and for each instant in the range we will count the number of rockets that contains this instant in their ranges by iterating over the given set of rockets and we will update the global variable with the maximum of current value of count and global variable.

**Follow the steps below to implement the above algorithm in C++:**

* Declare a variable **numberOfRockets.** It represents the number of rockets to be fired.
* Initialize a variable **maxEndTime** as 0. It stores the maximum end time out of all the rockets.
* Take **numberOfRockets** as input from the user.
* Declare a vector of pairs, **rockets**. It will hold a number of pairs.
* Iterate over from **index** = 0 to **index** = **numberOfRockets** –1using a for loop and at each iteration declare two variables, **startTime** and **endTime** and take these as an input from the user. These variables represent the instant at which rocket would leave the ground and the instant after which the rocket would be on the ground.
* Make a pair of **startTime** and **endTime** and insert this pair in the vector, **rockets** at the position, **index**.
* Simultaneously update the value of **maxEndTime** as the maximum of **maxEndTime** and **endTime**.
* After the end of for-loop, Initialize a variable **answer** as zero. It stores our final answer.
* Now iterate over **currentTimeInstant** = 0 to **currentTimeInstant** = **maxEndTime** using a for-loop, and initialize a variable **count** as zero. It represents the number of rockets that would be in the sky at the **currentTimeInstant**.
* Now run a nested range-based for-loop and iterate over the values of the vector, **rockets**.
* Increment the value of **count,** if **currentTimeInstant** lies between the first and second value of the pair.
* After the end of the ranged-based loop update the **answer** variable as the maximum of **count** and **answer**.
* Print the value represented by the **answer** variable.

Below is the implementation in C++:



[Source Code link C++](https://ideone.com/uI3y5l)

**Time complexity - O(maxEndTime \* N) where N is total number of rockets and maxEndTime is the**

**Maximum end time out of all the rockets**

**Auxiliary space - O(N) as space is required to store start time and end time.**

**Note that the naive approach would be able to pass partial test cases (constraint 1) only. For the constraint 2, It would result into time limit exceeded (TLE).**

**2. Efficient approach (Greedy algorithm):**

**Hint** : Think about sorting.

**Intuition** 💡 : Based on the constraints 2, we are required to think towards O(N) or O(N Log N) solution.

**Books on Microsoft Explanation : We can simulate with rocket start time and end time. Since a rocket will be in the sky at the instant of start time of the rocket so we can make a pair of start time and one, and insert it into an efficient data structure that can hold a number of pairs. Also the rocket would be on the ground at the instant just after the instant of end time. So again we would make a pair of end time + 1 and minus one, and insert it into the same data structure. Finally, we would be required to sort the data structure on the basis of first element of pairs.**

**Since, the constraints are big so we cannot sort this data structure using algorithms like bubble sort which can take O(N2) time and quick sort which can go upto O(N2) in worst case. We will have to use merge-sort since it takes O(N Log N) even in the worst case scenario. In C++, we can use inbuilt STL function sort() that is equivalent to merge-sort.**

**After sorting the data structure, now we will iterate over all the pairs in the data structure and keep the track of current rockets in the sky by simply adding the second value of the current pair and accordingly we can update our final answer on the go.**

**Follow the steps below to implement the above algorithm in C++:**

* Declare a variable, **numberOfRockets**. It represents the number of rockets are fired.
* Take **numberOfRockets** as input from the user.
* Declare a vector of pairs, **rockets**. It will hold a number of pairs.
* Iterate over from 0 to **numberOfRockets** –1using a for loop and at each iteration declare two variables, **startTime** and **endTime** and take these as an input from the user.
* Initialize two pairs, **start** and **end** as {**startTime**, 1} and {**endTime**, -1} respectively. Insert these pairs in the vector of pairs i.e., **rockets.**
* Sort the vector **rocket** on the basis of first value of pairs**.**
* After initializing the rocket vector with pairs, now initialize a variable **answer** as 0. It keeps the track of our final answer.
* Initialize another variable **current** as 0 and it represent current number of rockets in the sky.
* Now iterate over the vector **rockets** and update the value **current** by adding the second value of the pair. Also update the **answer** variable as the maximum of **answer** and **current**.
* Print the value represented by the **answer** variable.

**Below is the implementation in C++,**

****

[Source Code link C++](https://ideone.com/5vqiom)

**Time complexity - O(N Log N) where N is the total number of rockets.**

**Auxiliary space - O(N) where N is the total number of rockets as 2 \* N space is required to store start time and end time separately.**

**So by following the efficient approach we can pass both the constraints easily.**

**Test cases:**

|  |  |
| --- | --- |
| Test case 1:  10  114 972  393 852  192 243  799 973  204 510  126 661  356 360  627 877  639 961  14 354 | Test case 2:  6  349 854  403 618  116 865  450 525  137 164  66 505 |
| Test case 3:  3  258 309  131 701  656 928 | Test case 4:  5  68 449  319 329  285 452  428 858  38 264 |
| Test case 5:  15  594 788  19 186  43 500  321 440  588 967  25 334  492 572  723 734  567 809  91 872  154 176  475 604  539 811  310 827  128 631 | Test case 6:  2  267 604  448 905 |
| Test case 7:  5  61 96  34 122  340 774  15 799  374 887 | Test case 8:  11  903 960  99 980  372 855  284 553  228 663  73 834  355 392  279 696  23 138  465 907  560 904 |
| Test case 9:  6  73 435  100 565  354 705  280 360  176 872  818 895 | Test case 10:  7  39 173  639 675  482 972  520 923  39 233  8 277  47 838 |

**Solution file:**

|  |  |
| --- | --- |
| Solution 1:  5 | Solution 2:  5 |
| Solution 3:  2 | Solution 4:  3 |
| Solution 5:  8 | Solution 6:  2 |
| Solution 7:  3 | Solution 8:  7 |
| Solution 9:  5 | Solution 10:  4 |

**Some of my best works:**

DSA based articles: [Bit Manipulation](https://www.geeksforgeeks.org/count-of-subsequences-with-sum-two-less-than-the-array-sum/)

Other articles: [List of articles](https://auth.geeksforgeeks.org/user/bhuwanesh/articles)